proj - Project Cloner

Srini, rs@toprllc.com

October 19, 2020

1 Introduction

Prolific software developers have a cache of rich set of personal tools; leading to a need to develop new, experimental tools frequently. If for example **yaml** is a convenient file format for a configuration application but if I am not familiar with the toolset to manipulate these files, I would write some simple tools with experimentation in mind. Even large projects with many functions grow incrementally. Most disciplined software engineers have a pattern of software modules that works for them and want to replicate for all their efforts. This is true regardless of their preferred programming language.

In addition most modern development systems such as **nodejs** provide a way to perform an initial setup of a template project. The templates are usually complete, simple but somewhat tricky to setup.

In this projectlet, we develop a project cloner. Using a simple markup language, a template project can be cloned, customised and made available for further development.This is a rather ambitious effort requiring:

- a template project that is generic enough to clone, providing a reasonable base functionality.
- incorporate sufficient customizability and enable customization.
- if the above is programming language independent, we get the maximum benefit.
- the cloner app.

1.1 Learning Objectives

Configuration files stored in INI format In this application, the configuration needs are not too complex and thus we try a simpler format - compared to popular choices such as **json** or **yaml**.

Macro processing Building upon the ideas implemented in the **codemd** project, further explorations in processing source code interspersed with customization macros. Again the approach is programming language agnostic.

System Design More than the specific tools explored, this projectlet explores expanding the scope of the design effort, encompassing the template component and the tool. In particular the creation of a clonable component with just the right level of functionality will lead to gains in productivity.

2 Specifications

This command line tool creates a new project by cloning an existing template project and processing the sources executing the directives in each file resulting in a new project. The project to be cloned is specified by its **git** repository. The tool looks for a file of the name **Clone.cfg** at the root directory. The following is a simple example:

```
[project]
default=testcli
help=Name of the project to create
[copyright]
default=your name
help=amend the copyright statement with your name
```

In the above example, there are 2 **macros** that this project provides for customization. The macros can be overridden in the commandline of this tool. The macro processor can use the values supplied (or the default values) in the directives.

2.1 Processing directives

As a file is scanned line by line, proj searches for directives. When a line contains the keyword **proj:** the line is searched for directives and options. Directives and options are case sensitive.

Since proj ignores all other markers, it is language agnostic. Directives are included in line comments appropriate for the language (Eg. # for **Python** and **–** for **Ada**).

rename causes the file to be renamed in the target. The basename of the file is subsituted by the argument. For example:

```
// proj: rename $(Proj)
-- proj: rename $(Proj)_option
```

The first example line causes the file say **main.c** to **newproj.c** where newproj is the name of the new project being created. The second example will cause the same file to be renamed **newproj_option.c**

subs or substitute This causes the search for the specified word in a line to be substituted by the value of the macro. The macros may have default values specified in the configuration file. If the macro is not configured, and not supplied in the command line, the occurrence of the regular expression will be eliminated.

```
// proj: subs word $(Macro)
```

While scanning the line any part of the line after the occurrence of the keyword **proj:** is not processed.

The following meta macros are also processed by proj. Template projects will benefit from using these appropriately. These cannot be overridden in the command line.

```
# proj: $(Created)   -- Date and time of project creation
# proj: $(Username)  -- Username from the system or as overridden
```

2.2 How do we utilize

To summarize, this facility is then usable by anyone whatever be the programming language or environment. The only requirements are that you provide a **Clone.cfg** that lists some macros that are overridden each usage.

2.3 Potential Enhancements

Though it is reasonably complete and featureful to ensure certain uniformity in software, the tool has a lot of scope for additional features:

- Additional directives such as **if** will make this more powerful like the **C** preprocessor.
- Ability to clone partial structures will make it easier to have a uniform **C++** project component architecture; automatic generation of a class interface and implementation, unit test framework, a build environment like a **Makefile** will go a long way towards a smooth workflow.
- Repository - by populating a new repository for the newly created structure will eliminate expected next steps.

3 Implementation

3.1 Libraries

- glib library
- boost library - `https://www.boost.org/`

Implementation in **C++** `https://gitlab.com/cpp8/proj.git`
Implementation of an example template in **C++** `https://gitlab.com/cpp8/cli.git`